FT-1024 USER'S MANUAL

FORWARD TECHNOLOGY
INCORPORATED
2595 Martin Avenue
Santa Clara, CA 95050
(408) 988-2378
TWX 9103382186

TABLE OF CONTENTS

	Page #
INTRODUCTION	1
INSTALLATION	2
OPERATION OVERVIEW	4
PROGRAMMING	6
READING FROM THE 1024	10
PROGRAMMING EXAMPLE	12

TABLE OF FIGURES

		Page #
FIGURE 1	FT-1024 Graphics Controller	2
FIGURE 2	Device Address Switches	3
FIGURE 3	Video Connector Pinouts	3
FIGURE 4	Block Diagram of FT-1024	4
FIGURE 5	Frame Buffer	5
FIGURE 6	Programmable Registers	6
FIGURE 7	Address Bit Assignments	8
FIGURE 8	Read Sequence	11

FT-1024 SPECIFICATIONS

PHYSICAL:

WIDTH:	12.00 in. (30.48 cm)
HEIGHT:	6.75 in. (17.15 cm)
DEPTH:	.55 in. (1.40 cm)
WEIGHT:	1 lb
SHIPPING WEIGHT:	1.50 lbs
FORM FACTOR:	IEEE P-796

ELECTRICAL:

5 V ± 5% 4A (MAX)

ENVIRONMENTAL:

OPERATING TEMPERATURE:	0° C to 50° C
STORAGE TEMPERATURE:	0° C to 70° C
RELATIVE HUMIDITY:	90% WITHOUT CONDENSATION

INTERFACE:

Bus conforms to IEEE P-796
Pixel clock frequency: 32 MHz
Horizontal frequency: 32 KHz
Vertical frequency: 30 Hz
Separate TTL video and timing signal outputs
Optional ECL video and timing signal outputs

INTRODUCTION

The Forward Technology FT-1024 is a high performance IEEE 796 (Multibus®) compatible graphics controller designed specifically for systems which need high resolution black and white graphics combined with high speed pixel updates. The FT-1024 is designed to work in conjunction with Forward's FT-68M 68000 based 16-bit single board computer; however, it can be used with any 16-bit Multibus compatible CPU.

The FT-1024 overcomes one of the major problems associated with bit mapped graphics (the time required to manipulate the large numbers of bits in the frame buffer) by implementing in hardware several functions that have been done traditionally in software. This allows the FT-1024 to update up to 16 pixels per microsecond (a full frame in 64 milliseconds).

The FT-1024's frame buffer is 1024x1024 and is addressable using cartesian (x,y) coordinates. Location 0,0 is, by definition, in the upper left hand corner of the screen. Nominally, the video display is 800x1024 pixels, leaving 224x1024 pixels to store graphic symbols such as the cursor and character fonts.

The FT-1024 is designed to accomodate either the Ball HD series of CRT's or Motorola's M4408. Other high resolution interlaced monitors can be used as long as they require separate synchronization and video signals and require drive signals of the frequencies listed in the FT-1024's specifications.

INSTALLATION

The FT-1024 is available in two versions. These versions are identical except the -01 version has an ECL video output driver and a connector at location J1, whereas the -00 version's video connector is a J2 and provides TTL level outputs. The FT-1024 is designed to plug into any standard Multibus® or P-796 bus. The FT-1024 can be configured to reside in any one of eight different memory locations. The beginning address of the FT-1024 is set by the device address

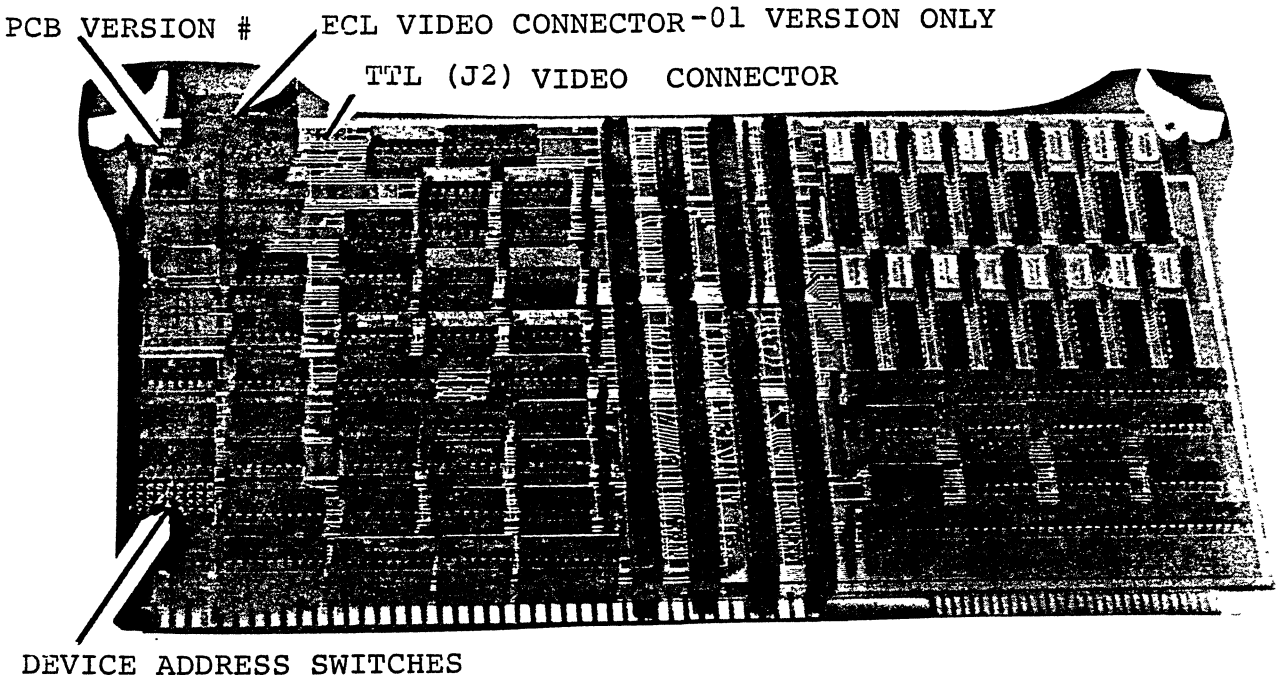


FIGURE 1: FT-1024 Graphics Controller

switches (see Figure 1 for switch locations). Figure 2 defines these switch settings.

SW# 'ON'	STARTING ADDRESS
1	x 'E0000'
2	x 'C0000'
3	x 'A0000'
4	x '80000'
5	x '60000'
6	x '40000'
7	x '20000'
8	x 'ØØØØØ'
Note: Only 1 switch may be on at a time	

FIGURE 2:
Device Address Switches

It is the customer's responsibility to fabricate the necessary cabling to connect the FT-1024 to the CRT. The video and synchronization signals are available at connector J2 (on the -00 version) and at connector J1 (on the -01 version). Figure 3 defines the pin outs for each connector. Both J1 and J2 use a 10-pin right angle header. The

SIGNAL NAME	J2 PIN #	J1 PIN #(ECL)
$\overline{\text{VIDEO}}$	3	1
VIDEO	5	2
HSYNC	7	6
$\overline{\text{HSYNC}}$	NA	5
VSYNC	9	8
$\overline{\text{VSYNC}}$	NA	7
GND	1,2,4,6,8,10	3,4,9,10

FIGURE 3:
Video Connector Pinouts

matching connector for this header is an SAE CP6310BRS.

OPERATION OVERVIEW

Figure 4 is a block diagram of the FT-1024. The FT-1024 utilizes a frame buffer, a source register, a mask register, and a set of command registers (function, width, control and interrupt) and four sets of x & y address registers in addition to a function unit to control display data sent to the frame buffer. Data from the frame buffer or the data bus is used to load the source and mask registers. Data from the frame buffer is sent to the video circuitry where it will be serialized for use by the CRT.

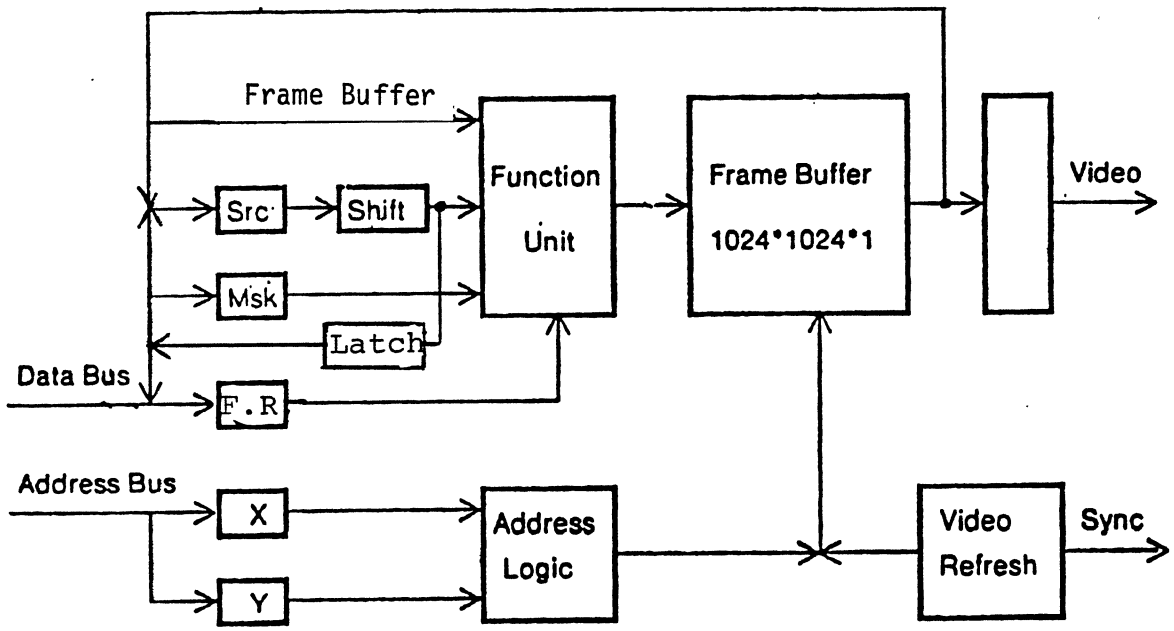
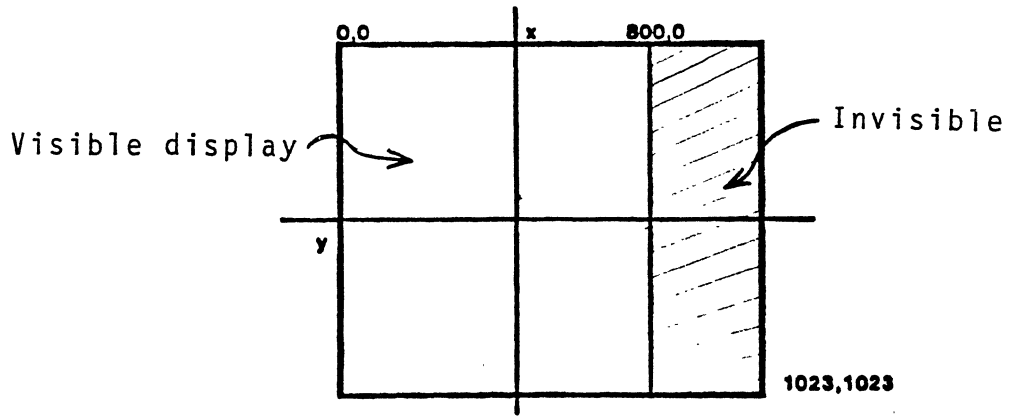


FIGURE 4: Block Diagram of FT-1024

The frame buffer is divided into 1024x1024 locations. Each location represents a single pixel on the CRT. Each pixel is addressable and can be modified using one of the 256 raster operations (rasterops) which perform bit-wise logic functions on the data contained in the frame buffer and the source and mask registers. The host processor accesses each pixel within the frame buffer via one of the x & y register pairs. The four pair of x,y registers allow the programmer to maintain simultaneous pointers to up to four graphical objects using FT-1024 hardware. Coordinate location Ø,Ø is, by definition, the upper left hand corner of the CRT. From one to 16 pixels may be read from or written to the frame buffer in one cycle (one microsecond).



Size: 1024 * 1024 pixels
Visible: 800 * 1024 pixels
Invisible: 224 * 1024 pixels
Updates: 16 pixels/cycle (max)

FIGURE 5: Frame Buffer

The FT-1024 can be programmed to provide an interrupt every vertical blanking interval so that the software and the display update can be synchronized. The interrupt level is also programmable; that is, the programmer determines which of the eight bus interrupt signals will be generated (BINTØ-BINT7). This allows the programmer to control the priority the FT-1024 will have on the bus.

PROGRAMMING THE FT-1024

The programmer has four different registers he can access and modify in order to produce a display on the CRT. (See Figure 6).

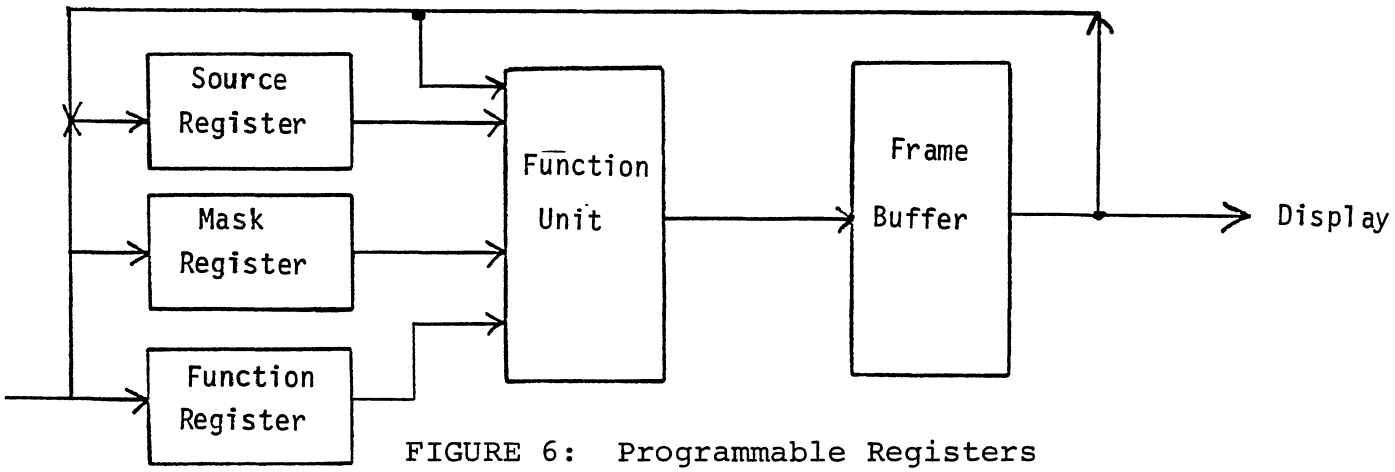


FIGURE 6: Programmable Registers

The FT-1024 is initialized by either clearing (all Øs) or setting (all 1s) the frame buffer. Once the programmer has done this, any change to the display (i.e., pattern generation) is done by loading the source register with the data to be displayed, loading the mask register with the appropriate pattern information, and then loading the function register with an opcode derived from the resultant of the type of boolean operation desired. The programmer can perform the following boolean functions between any combination of the frame buffer, the mask register, and the source register: OR, NOR, XOR,

NXOR, AND, NAND, and DO NOTHING. To accomplish this boolean function, the CPU must do a like function and load the resultant opcode into the FT-1024's function register. The FT-1024 will then perform the boolean function in its function unit, and load the resultant into the frame buffer. The frame buffer contents will then be displayed. The purpose behind being able to do all of these boolean functions (256 combinations) is to allow the programmer to merge different pieces of data in such a manner as to create realistic images on the display.

There is a unique code associated with the frame buffer, the mask register, and the source register. These codes are:

Frame Buffer Only	x 'AA'
Mask Register Only	x 'FØ'
Source Register Only	x 'CC'
Clear Frame Buffer	x 'ØØ'
Set Frame Buffer	x 'FF'

By selecting one of these codes and loading it into the function register the appropriate register or buffer will be accessed and its contents will be placed in the frame buffer at the location addressed by the x & y registers.

If the programmer desires to perform a boolean function (such as "AND") between two registers, all that is necessary is to perform the "AND" function between the code associated with the registers and load the resultant opcode into the function register. For example: to "AND" the frame buffer contents with the source register contents, the programmer would "AND" x 'AA' and x 'CC' and send the resultant opcode (x '88') to the function register.

Selecting the frame buffer only (x 'AA') will result in a DO NOTHING; as the hardware will simply rewrite the current contents of the frame buffer back into the frame buffer.

The FT-1024 PCB decodes 20 of the IEEE P-796 address lines. Each line is assigned a specific function, as indicated in Figure 7. Up to eight FT-1024's can reside on the same bus. Each FT-1024 must have a unique starting address (See Figure 2 for details).

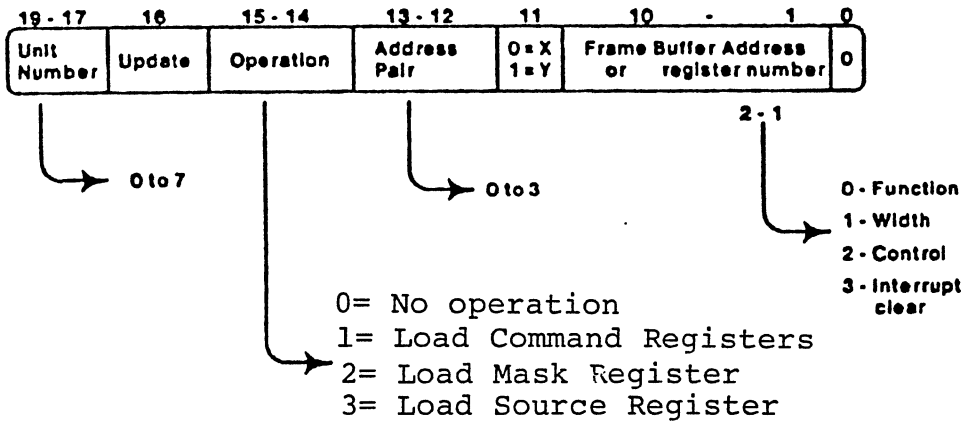


FIGURE 7: Address Bit Assignments

Figure 7 defines the 20 address lines which are decoded and identifies the meaning assigned to each bit. The following text describes these bit assignments:

BITS	FUNCTION
17-18-19	These address lines are decoded to determine which FT-1024 is being accessed.
16	This bit, when set (1) will allow the frame buffer to be updated (modified). When zero (0) the programmer can update the command registers without affecting the frame buffer.
14-15	These bits select which registers will be updated.

	15	14	
	0	0	-- No operation (x and y registers will be loaded)
	0	1	-- Load command registers (as defined by address bits 1 and 2)
*	1	0	-- load source register
*	1	1	-- load mask register (the number of bits loaded is determined by the width register)

* NOTE: If these registers are selected during a processor write, the data loaded will come from the data bus. If they are selected during a processor read, the data loaded will come from the frame buffer.

12-13	These bits are used to select which x or y register will be loaded. There are 4 x and 4 y registers. On any and every operation an x or y register will be updated.
11	Determines if the register identified by bits 12 and 13 is an x or y register. Ø = x, 1 = y.
1-10	These bits define the address within the frame buffer that data is to be sent to. If bits 14 and 15 define a "load command registers" operation, bits 1 and 2 will be used to define which command register is to be loaded.

Bit 2	1	Selected Register
0	0	Function register
0	1	Width register
1	0	Control register
1	1	Clear interrupts

The width register specifies the number of bits which can be loaded into the source register (Ø-15) and the number of pixels which will be updated in the frame buffer. The control register is used to enable interrupts (bit 8), set the interrupt level (bits 13, 14, and 15) and enable

the video output (bit 9). If interrupts are enabled the FT-1024 will generate an interrupt every vertical blanking interval so that software can be synchronized with the display update.

By using bits 13, 14, and 15 the programmer can specify which interrupt the FT-1024 will generate (B INTØ-B INT7).

BUS BIT	15	14	13	INTERRUPT GENERATED
	0	0	0	BINTØ
	0	0	1	BINT1
	0	1	0	BINT2
	0	1	1	BINT3
	1	0	0	BINT4
	1	0	1	BINT5
	1	1	0	BINT6
	1	1	1	BINT7

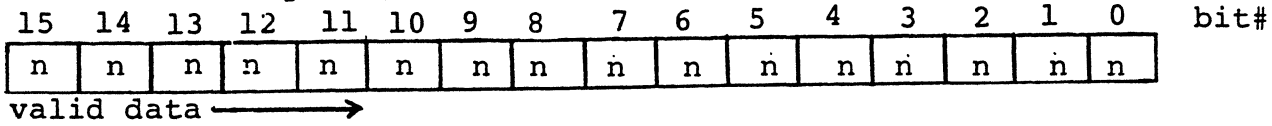
Pending interrupts can be cleared by having both bits 1 and 2 active (1's).

Ø Bit Ø should always be a Ø.

READING FROM THE FT-1024

The FT-1024 provides the programmer with the ability to read information from the source register and presents this information to the multibus. The programmer can load the source register with the contents of the frame buffer in order to facilitate a frame buffer read. The programmer can also read the mask register; it is necessary to load the mask register contents into the frame buffer and then into the source register.

The information sent to the multibus is high-order bit justified. The "word length" of the information reaching the bus is determined by the width register (i.e., if the width was set for 16 bits, all 16 bits of the information would be valid; if the width were set at 1 only bit, 15 would be valid).



Valid data bits determined by width register

Because of hardware constraints, reading information from the FT-1024 is a two-step process. See Figure 8.

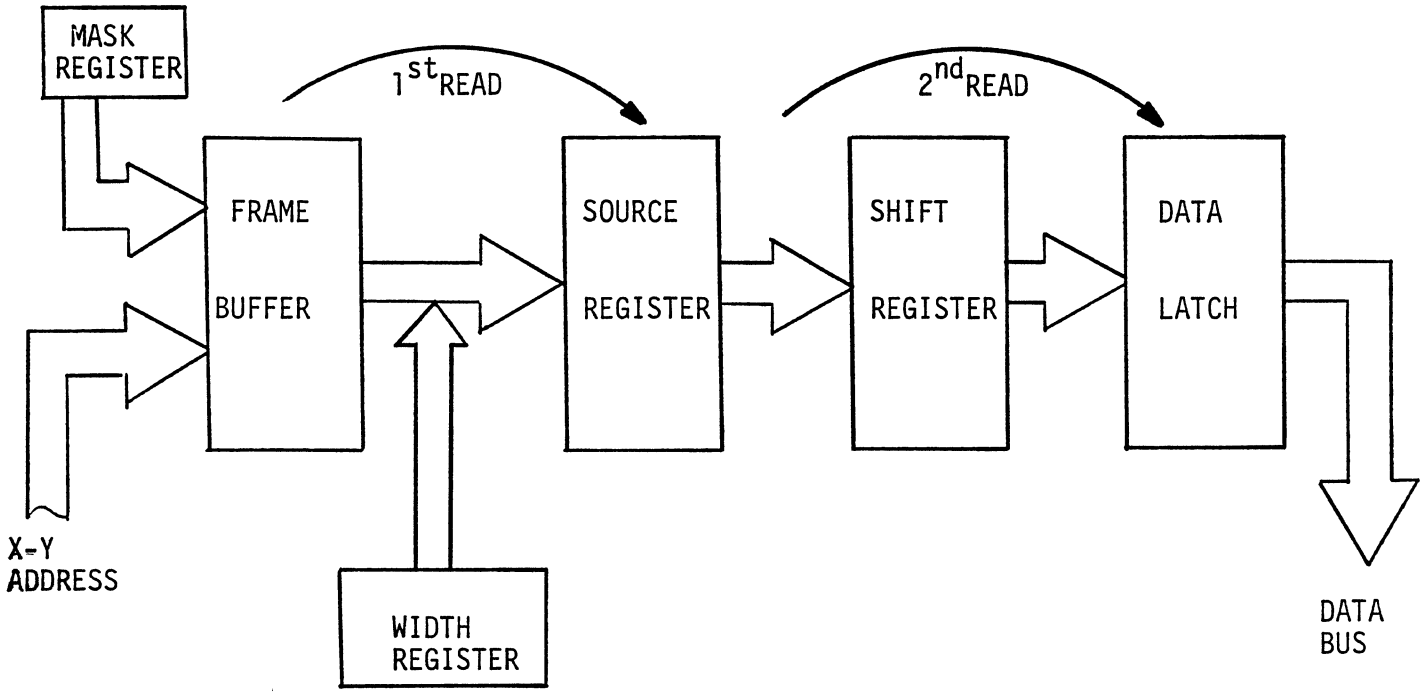
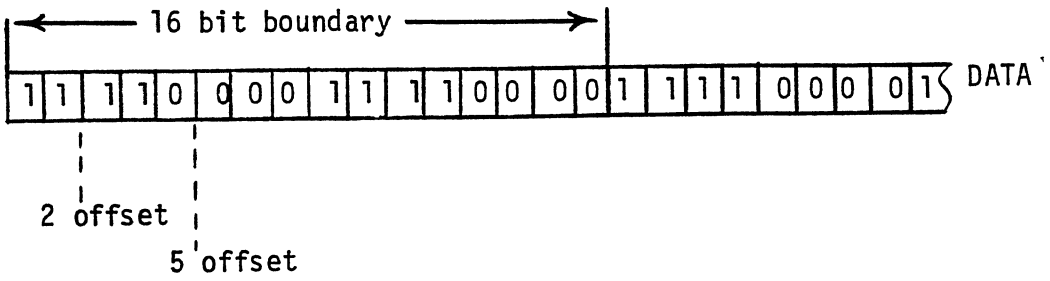


FIGURE 8 FT-1024 READ SEQUENCE

Data to be read from the addressed XY location in the frame buffer is first loaded into the source register by doing a read operation (the number of valid data bits is determined by the width register). An additional read operation must be performed to load the contents of the source register into the data latch. The contents of the data latch are made available to the data bus. Note that the data latch will always contain the contents of the source register which was loaded on the previous read cycle. The data latch is, in effect, 1 step behind the current read cycle.

When reading on other than 16 bit boundaries, the shift register (see Figure 8) automatically "justifies" the data in the data latch. The following example illustrates this:



* When read on the first 16 bit boundry , data = FØFØ
* When read off-set by 2, data = C3C3
* When read off-set by 5, data = 1E1E

* Assumes width register is set to 16

PROGRAMMING EXAMPLE

GRAPHMACS.h

```
#include "graphics.h"

#define GXBITMAPSIZE (1024) /*length of side of bitmap */

#ifndef XBIAS
#define XBIAS  (150) /*x-offset from edge of screen to (0,0) */
#endif XBIAS

#ifndef YBIAS
#define YBIAS  (256) /*y-offset from edge of screen to (0,0) */
#endif YBIAS

#define SHOWPOINT   GXset
#define ERASEPOINT  GXclear

/*
 *POINT(x,y) sets the given point to whatever the prevailing
 *function is.
 */
#define   POINT(x,y){\
  GXsetX((x)+XBIAS);\
  *(short*)(GXUnit0Base|GXupdate|GXsource|GXselectY|(((y)+YBIAS)<<1))=\
                                                    0xFFFF;\
 }
/*
 *SETGXFUNC(f)sets the function register to f
 */
#define SETGXFUNC(f)  GXfunction=(f);
```

```
/*
 *SETGXWIDTH(w) sets operation width to w
 */
#define SETGXWIDTH(w) GXwidth=(w);

/*
 *SETPOINT(x,y) turns on the point at (x,y)
 */
#define SETPOINT(x,y){\
  SETGXFUNC(SHOWPOINT);\
  POINT(x,y);\
 }
/*
 *CLEARPOINT(x,y) turns off the point at (x,y)
 */
#define CLEARPOINT(x,y){\
  SETGXFUNC(ERASEPOINT);\
  POINT(x,y);\
 }
```

Rastercopy.c

```
#include "graphmacs.h"


Rastercopy(xsrc,ysrc,h,w,xdst,ydst,funct,erase)
  register short            h,w;     /*d7, d6 */
  short                     xdst,xsrc,ydst,ysrc,funct
  char                      erase;
{
  short          temp;
  register short                                /*a5,4,a3,a2*/

          *Y_Dst_Address=
       (short*)(GXUnit0Base|GXsource|GXupdate|GXselectY+(ydst<<1)),
          *X_Dst_Address=(short*)(GXUnit0Base|G selectX+(xdst<<1)),

/*
   Temporarily, we will not use a second address cache.

*/
          *Y_Src_Address=
            (short*)(GXUnit0Base|GXselectY+(ysrc < 1)),
          *X_Src_Address=
            (short*)(GXUnit0Base|GXselectX+(xsrc < 1)),

   register                 i,j;          /*d5,d4 */

          GXfunction = funct;             /*Move(Source Reg)to dest. */
```

```
        /* If erase == 1 then Erase the source after Reading it */
/*      if (erase){
           GXfunction = ERASEPOINT;
           Y_Src_Address+=GXupdate;
        }
/*      /* Precompute iteration for both the x and y loops */
        GXwidth = 16;

        for (i=w/16; i>0 ; i--){

      /* Do the first read now to get the pipeline started */
       *X_Src_Address=1;
      temp=*YXSrc_Address++;

          for (j=h; j>0; j--){
        *X_Src_Address=1;
           temp=*Y_Src_Address++;

        *X_Dst_Address = 1;
           *Y_Dst_Address++ = temp;

             Y_Dst_Address-=h;
             Y_Src_Address-=(h+1);
             X_Dst_Address+= 16;
             X_Src_Address+= 16;

       }


      /* if there was any leftover width perform one special iteration */
      if (y &=0xF){
         0Xwidth = w;                          /*set a narrower width */

*X_Src_Address =1;
temp = *Y_Src_Address++;

     for (j=h; j>0; j--){
  *X_Src_Address=1;
     temp = *Y_Src_Address++;
  *X-Dst_Address =1;
      *Y_Dst_Address++ = temp;
```